RSAC 2025 Conference

Many Voices. One Community.

SESSION ID: USENIX-M05B

ENG25519: Faster TLS 1.3 handshake using optimized X25519 and Ed25519

Çetin Kaya Koç

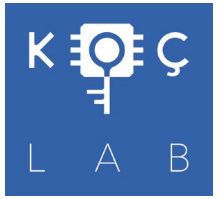
IEEE Life Fellow; Co-founder of CHES conference; Founder of Journal of Cryptographic Engineering
University of California Santa Barbara, NUAA and Iğdır University
https://cetinkayakoc.net

Disclaimer

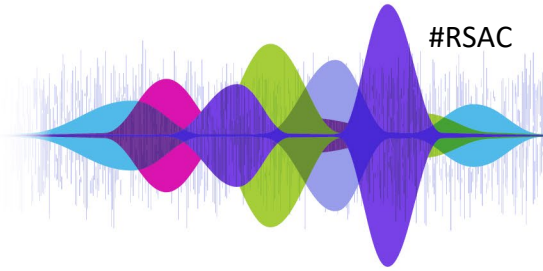
Presentations are intended for educational purposes only and do not replace independent professional judgment. Statements of fact and opinions expressed are those of the presenters individually and, unless expressly stated to the contrary, are not the opinion or position of RSA Conference LLC or any other co-sponsors. RSA Conference LLC does not endorse or approve, and assumes no responsibility for, the content, accuracy or completeness of the information presented.

Attendees should note that sessions may be audio- or video-recorded and may be published in various media, including print, audio and video formats without further notice. The presentation template and any media capture are subject to copyright protection.

© 2025 RSA Conference LLC or its affiliates. The RSAC and RSAC CONFERENCE logos and other trademarks are proprietary. All rights reserved.



ENG25519: Faster TLS 1.3 handshake using optimized X25519 and Ed25519



Jipeng Zhang[1], Junhao Huang[2,3], Lirui Zhao[1], Donglong Chen[2], Çetin Kaya Koç[1,4,5]

[1]Nanjing University of Aeronautics and Astronautics

[2]Guangdong Provincial Key Laboratory IRADS, BNU-HKBU

[3]Hong Kong Baptist University

[4]Igdır University [5]University of California Santa Barbara

Artifact: https://github.com/Ji-Peng/eng25519_artifact

Usenix Security 2024 Distinguished Paper Award

Outline

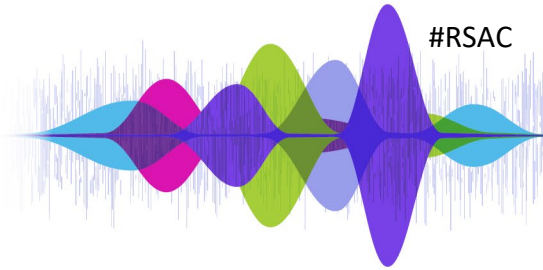
- Motivations
- Background
- Optimized X/Ed25519 implementation & ENG25519
- Conclusions
- “Apply”

RSAC 2025 Conference

Motivations

Speeding up TLS Cryptography

Many Voices.
One Community.



Motivations

- How can AVX-512IFMA instructions accelerate ECC?
- How can the optimized ECC implementation be integrated into TLS applications?
- How can the cold start issue of vector units be mitigated?
- To what extent can our optimized cryptographic implementation improve TLS applications?

RSAC 2025 Conference

Background

AVX-512, X/Ed25519 and TLS 1.3

Many Voices.
One Community.

Background: AVX-512

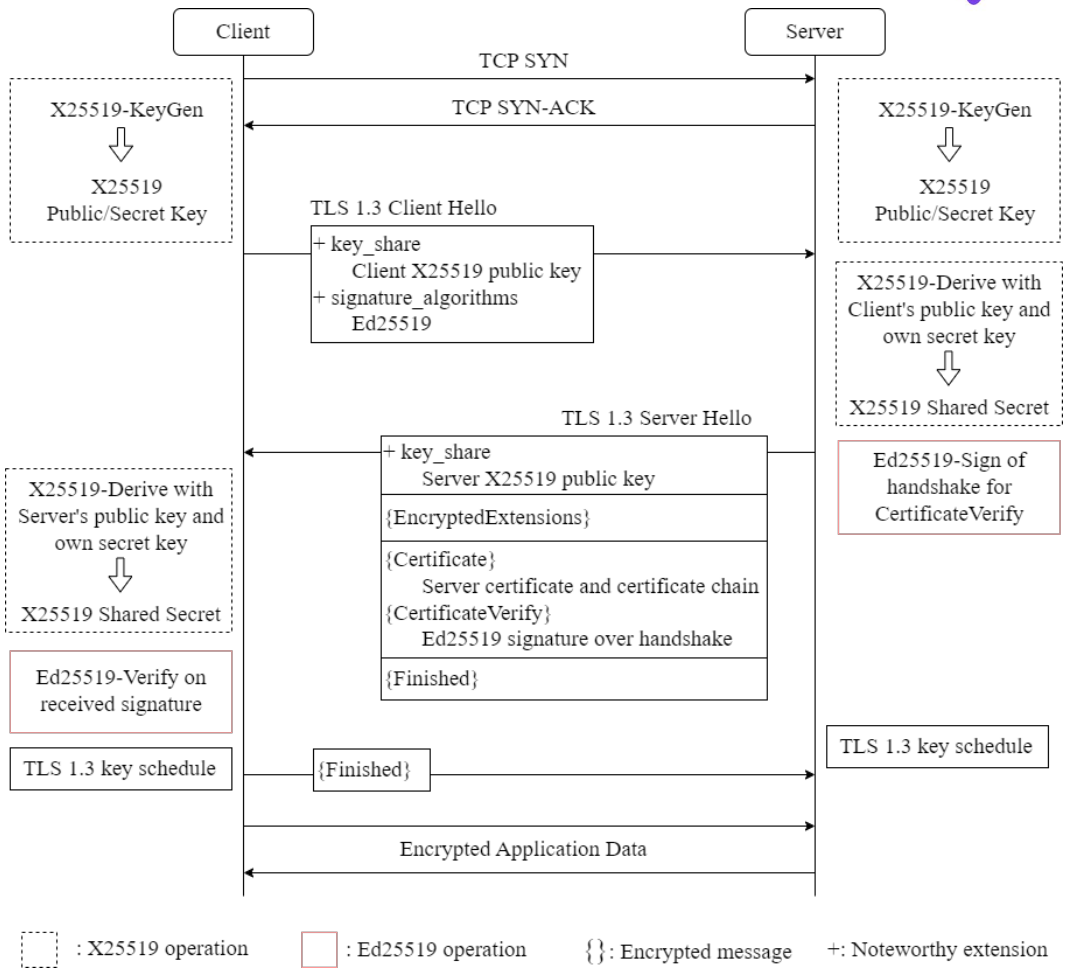
- AVX-512
  - 32 512-bit registers; Each 512-bit register can be divided into 32 16-bit, 16 32-bit, or 8 64-bit segments.
  - AVX-512IFMA supports **52-bit multipliers**, whereas AVX2 and AVX-512F only support 32-bit multipliers.

Background: X25519 and Ed25519

- X25519 and Ed25519
  - X25519, designed by Daniel J. Bernstein, is a Diffie-Hellman key exchange protocol based on Curve25519.
  - Ed25519, designed by Daniel J. Bernstein et al., is an Edwards-curve digital signature algorithm.
  - In 2018, RFC 8446 included X25519 and Ed25519 in the supported cipher suites for TLS 1.3.

Background: TLS 1.3 & DNS over TLS

- TLS 1.3 handshake
  - Client op; Server op.
  - X25519-KeyGen/KeyGen+X25519-Derive+Ed25519-Sign+X25519-Derive+Ed25519-Verify.
- DNS over TLS
  - TLS handshake → DNS queries and responses over the TLS connection.



TLS 1.3 handshake

RSAC 2025 Conference

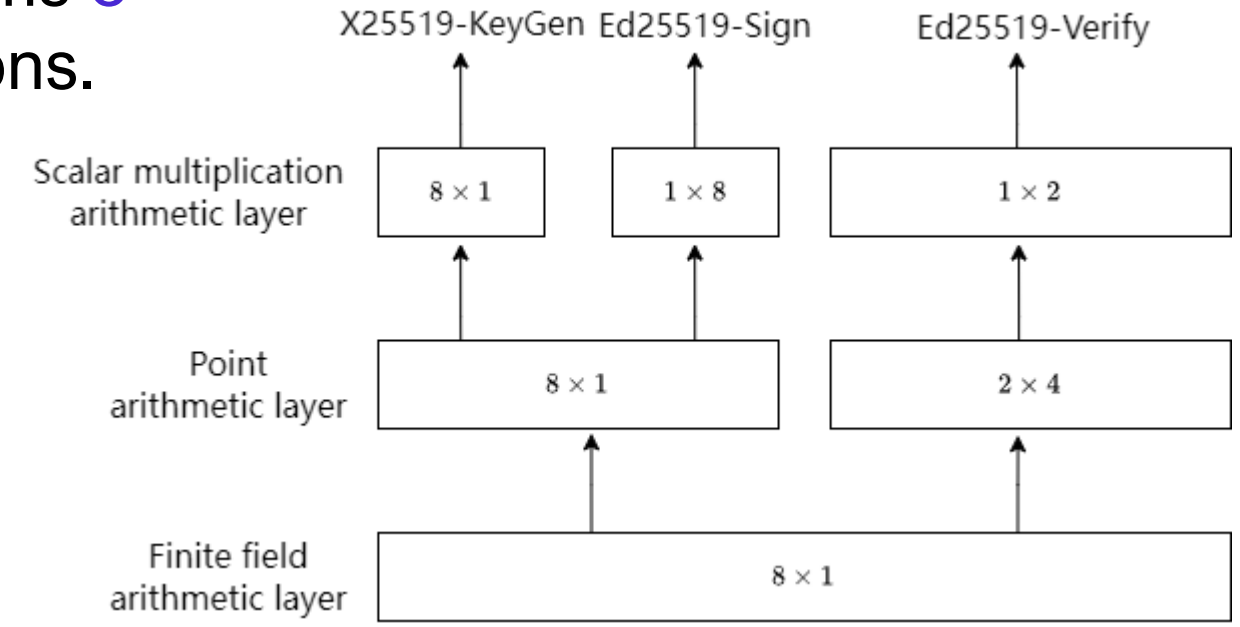
Optimized X25519 and Ed25519 implementation

Efficient Arithmetic

Many Voices.
One Community.

Optimized X/Ed25519 implementation

- Field arithmetic
  - Radix-2^{51}: A field element $f = f_0 + 2^{51} f_1 + 2^{102} f_2 + 2^{153} f_3 + 2^{204} f_4$
  - 8×1-way: One subroutine performs 8 parallel independent field operations.
  - We formally verified our field implementations using CryptoLine.



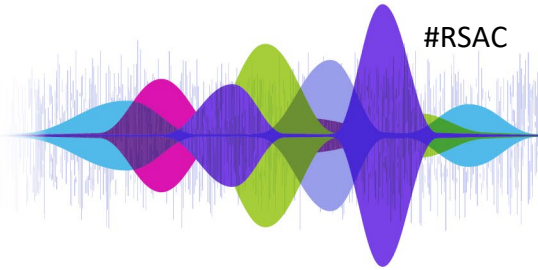
An overview of our implementation

Optimized X/Ed25519 implementation

Strategy: “finite field arithmetic” → “point arithmetic” → “scalar multiplication”

- X25519-KeyGen: $8 \times 1 \rightarrow 8 \times 1 \rightarrow 8 \times 1$
  - 12 times that of the OpenSSL impl. and 2.32 times that of Cheng et al.'s impl.
- X25519-Derive: We don't provide a faster X25519-Derive implementation than Hisil et al.
- Ed25519-Sign: $8 \times 1 \rightarrow 8 \times 1 \rightarrow 1 \times 8$
  - 3.79 times that of the OpenSSL impl. and 1.18 times that of Faz-Hernández et al.'s impl.
- Ed25519-Verify: $8 \times 1 \rightarrow 2 \times 4 \rightarrow 1 \times 2$
  - 3.33 times that of the OpenSSL impl. and 1.33 times that of Faz-Hernández et al.'s impl.

ENG25519: An OpenSSL ENGINE

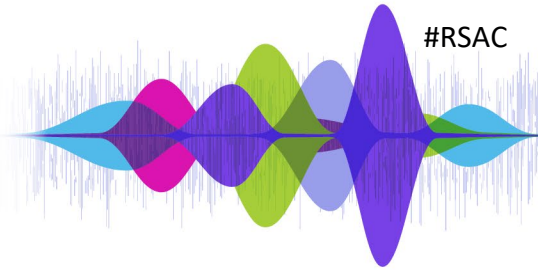


- ENG25519 is based on OpenSSL ENGINE APIs, libsuola, and engntru.
- Our optimized X/Ed25519 implementations can be transparently integrated into OpenSSL and TLS applications through ENG25519.

Detailed configuration of ENG25519.

Subroutine	Implementation
X25519-KeyGen Ed25519-KeyGen	Our $8 \times 1 \rightarrow 8 \times 1 \rightarrow 8 \times 1$ impl. $batch\text{-}size = 16$
X25519-Derive	$4 \times 2 \rightarrow 1 \times 4$ impl. of Hisil et al.
Ed25519-Sign	Our $8 \times 1 \rightarrow 8 \times 1 \rightarrow 1 \times 8$ impl.
Ed25519-Verify	Our $8 \times 1 \rightarrow 2 \times 4 \rightarrow 1 \times 2$ impl.

How to mitigate the cold-start issue?



- The processor will set the upper parts of the AVX2/AVX-512 vector units to a low-power mode to save power if the units are not in use for about 675 μs, leading to a warm-up phase of approximately 14 μs (56,000 clock cycles at 4 GHz) when an AVX2/AVX-512 instruction is executed in the low-power mode.
- During the warm-up phase, the throughput of the related instructions is 4.5 times slower than usual.
- All X/Ed25519 primitives suffer from varying degrees of performance degradation; especially the X25519-KeyGen takes 3.8 times longer in the DoT scenario than in the warm-start scenario.

How to mitigate the cold-start issue?

- We designed a heuristic auxiliary thread that performs different actions based on the application's varying load conditions.
  - Low-load scenarios: It takes no action to avoid disrupting the processor's power-saving strategies.
  - Medium-load: It periodically executes a vector instruction.
  - High-load: The frequent cryptographic operations inherently prevent entering low-power mode.

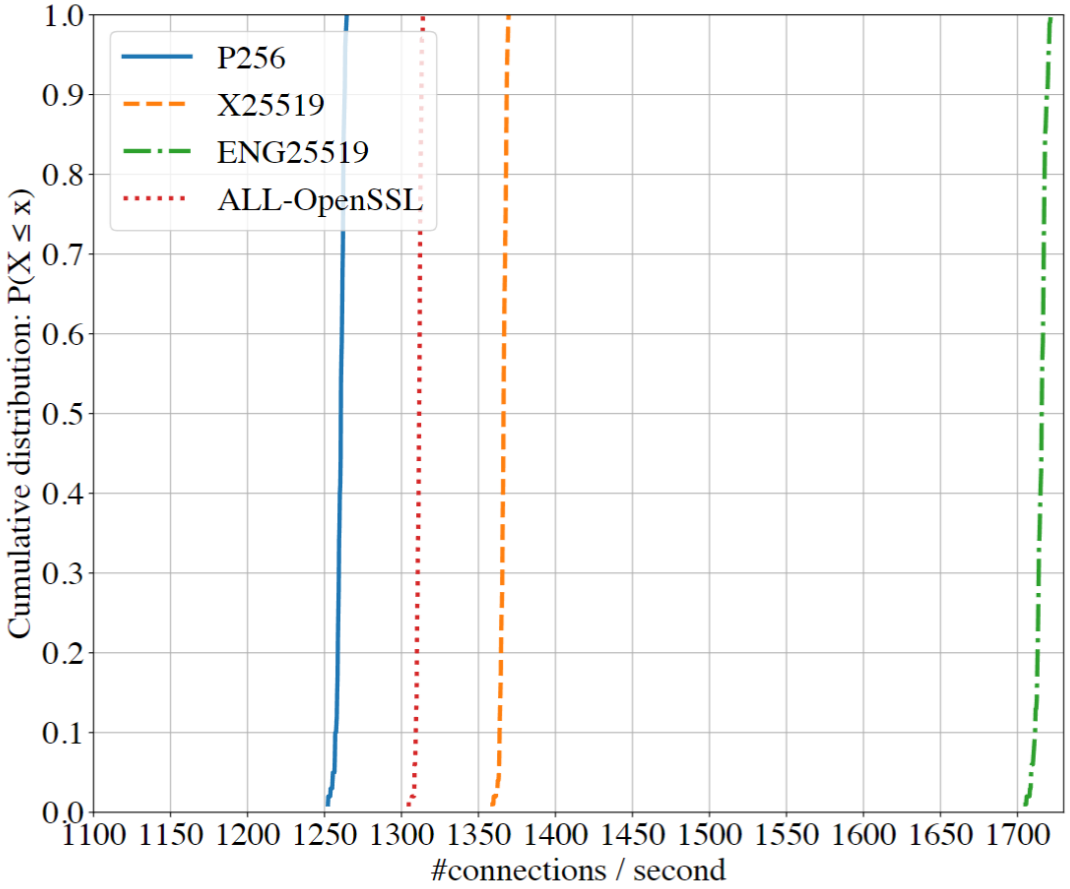
Amortized CPU cycles (CC) to generate a keypair.

Batch size	Amortized CC with auxiliary thread	Amortized CC without auxiliary thread
1	10,315	28,450
4	9,107	19,388
8	9,003	14,108
16	8,980	11,406

Benchmark of TLS handshake

Client: tls_timer ↔ Server: OpenSSL s_server

On average, the proposed ENG25519 setting (1,707 #connections/second) enables 25% and 35% more handshakes per second than X25519 (1,366) and P256 (1,260), respectively.



Benchmark of DoT query

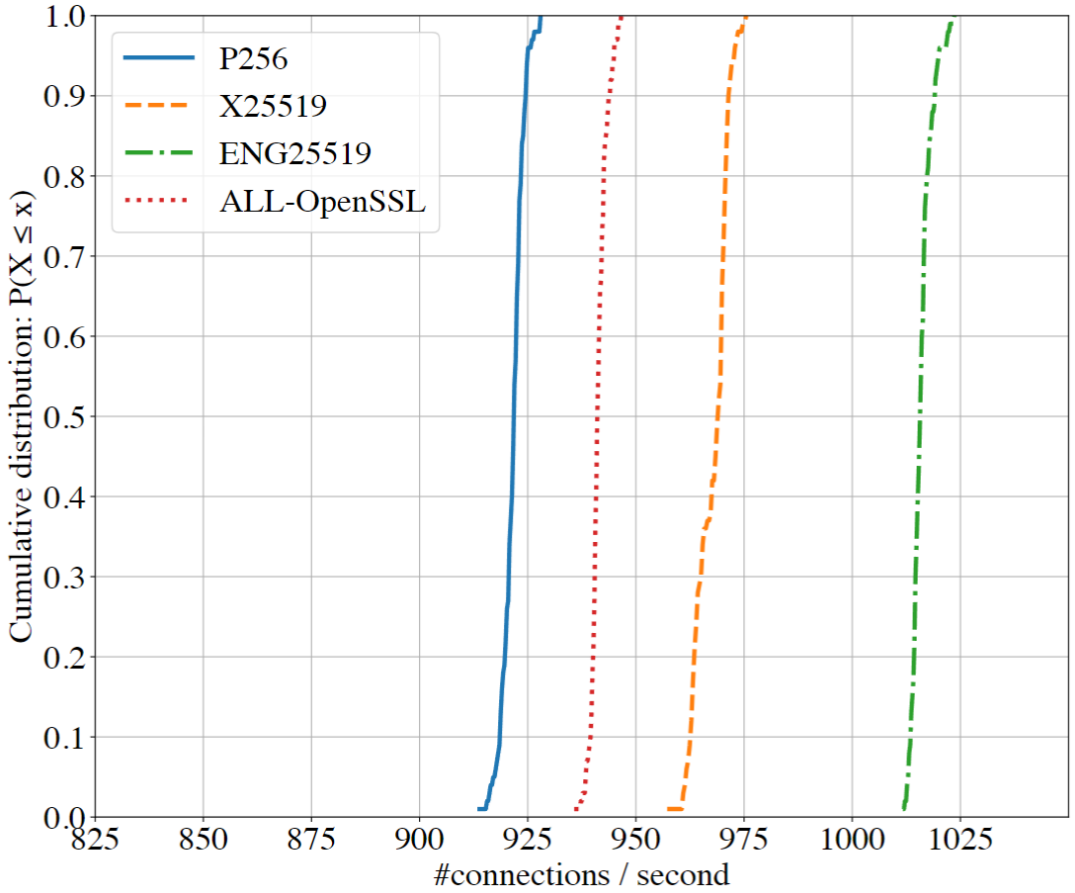
Client: dot_timer↔ Server: unbound DoT server

End-to-end experiments:

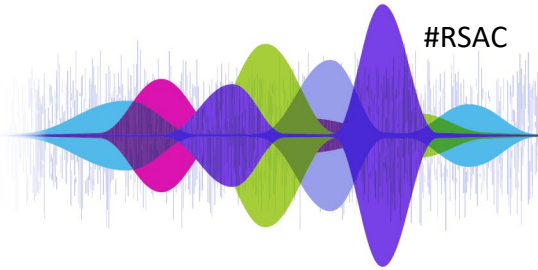
- Our ENG25519 outperforms all other configurations.

Peak throughput:

- Our ENG25519 achieved a 41% and 24% increase over P256 and X25519, respectively.



Conclusions



- Faster X/Ed25519 implementation using AVX-512IFMA.
- Integration of optimized X/Ed25519 implementations into TLS; faster TLS 1.3 handshake; increased DNS over TLS throughput.
- Under cold start conditions, some primitives may suffer a performance degradation of up to 3.8 times. If the vector implementation does not achieve significant improvements, a reevaluation of the vector implementation versus the x64 implementation is necessary.
- Open-source artifact: https://github.com/Ji-Peng/eng25519_artifact.

RSAC 2025 Conference

Apply What You Have Learned Today

What are the next targets for research?

Many Voices.
One Community.

Apply What You Have Learned Today

- For the short-term we need to explore:
  - Why are CPU vendors implementing matrix computation instruction set extensions? Examples include Apple's AMX, Intel's AMX, and ARM's SME. What drives this trend?
  - Investigating the importance of formal verification across research domains. Why is formal verification gaining traction in various fields?
- For the long-term we need to explore:
  - Are there computations in your field of interest that could benefit from SIMD or matrix extensions? Consider whether certain algorithms or workloads could be optimized using these hardware acceleration techniques.

RSAC 2025 Conference

Many Voices.
One Community.

Thanks

cetinkoc@cryptocode.com